Öğr.Gör. Mustafa GÖKMEN
gokmen@selcuk.edu.tr

Nesneye Yönelik Programlama

Data Types

Java Veri Tipleri

Data Types								
Primitive								Non-Primitive
Boolean	Numeric							Compound Types ✓ Array ✓ ArrayList ✓ Vector ✓ String etc.
	Character (Unicode)	Integral						
		Integer				Decimal (IEEE 754)		
boolean	char	byte	short	int	long	float	double	
Boolean	Character	Byte	Short	Integer	Long	Float	Double	Wrapper Classes
false	'\0'	0	0	0	0L	0.0f	0.0d	null
1 Bit	2 Byte	1 Byte	2 Byte	4 Byte	8 Byte	4 Byte	8 Byte	Mutative

Primitive türlerin ***kullanılmadan önce*** mutlaka bir değeri olmalıdır. Ancak Non-Primitive türlerin bir değeri olmak zorunda değildir ve belirtilmemiş ise ***null*** olacaktır.

Referans Tipler (İşaretçiler)

- `new` operatörü ile bir nesne oluşturulduğunda bu nesnenin bellekteki adresi döndürülür ve bu adres, oluşturulan nesneye daha sonra erişmek amacıyla referans olarak kullanılır. Bu referansı saklayabilmek için de bir değişkene (Ör: `myObj`) gereksinim vardır.

```
Class myObj = new Class();
```

- İçerisinde referans (nesne adresi) saklayabilen türden değişkenlere Referans değişken veya Nesne Tutamacı (Object Handle) adı verilir ve varsayılan değer olarak `null` değerini alırlar. `null`, bir nesne tutamacının içinde o anda herhangi bir nesnenin adresi olmadığını belirten değerdir.
- `new` ile yeni bir nesne oluşturulduğu zaman, JVM bu nesneyi heap adı verilen global bellek alanında saklar. Nesnenin adresi ise stack adlı yerel bellekteki bir referans değişkenin içinde saklanır.
- Bu durumda, nesneyi oluşturan kod satırının içinde bulunduğu metod sona erdiğinde nesnenin referansını saklayan değişken stack alanından atılır. Ancak nesnenin kendisi hala heap bellektedir. Bir Java programının çalıştığı süre boyunca oluşturulan bütün nesneler, heap alanında saklanır.
- Çöp Toplayıcı (Garbage Collector), yığında bulunan ve kendisine hiç referans olmayan nesneleri (ki bunlara çöp denilebilir) bularak bunları ortadan kaldırır ve bunların kullanmakta oldukları bellek alanlarını yeniden yığına kazandırır.

Sarmalayıcı Sınıf (Wrapper Class)

Wrapper Class (Sarmalayıcı Sınıf); temel veri türlerinin birer nesne olarak kullanılabilmesini sağlar. Bir `int` tipi sayı sadece bir tek tamsayı değerini taşıyabilirken, Sarmalayıcı Sınıf tipinde bir `Integer` nesne tanımlanırsa, temel veri tipi (`int`) ile birlikte o tipe ait yeni metodlar ve yeni özellikler de barındırabilen bir nesne elde edilmiş olacaktır.

Temel veri tipleri ile sabit boyutlu diziler oluşturulabilirken, Sarmalayıcı tipte veriler ile listeler oluşturulabilir. *Liste oluşturmak için temel veri tipleri kullanılamaz*.

```
ArrayList<int>     arrInt = new ArrayList<>(); // Invalid
ArrayList<Integer> arrInt = new ArrayList<>(); // Valid
```

Temel Veri Tipleri	Referans Veri Tipleri
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Sarmalayıcı Sınıf (Wrapper Class)

```java
public class AutoBoxing {
  public static void main(String[] args) {
    Integer intNumber = 123;     // Autoboxing
    Double doubleNumber = 5.72; // Autoboxing
    Character charNumber = 65;  // Autoboxing & Implicit cast
    int i = intNumber;          // Unboxing

    System.out.println(intNumber);
    System.out.println(doubleNumber);
    System.out.println((char) charNumber); // Explicit Cast
    System.out.println("----");

    // The following regular forms do the same above
    System.out.println(intNumber.intValue());
    System.out.println(doubleNumber.doubleValue());
    System.out.println(charNumber.charValue());
    System.out.println("----");

    System.out.println(doubleNumber.toString().length());
  }
}
```

```
123
5.72
A
----
123
5.72
A
----
3
```

Nesne Atamaları

Nesneler için atama işlemleri, temel tiplere göre daha karmaşıktır. Nesneleri yönetmek için referanslar kullanılır. Eğer, nesneler için bir atama işlemi söz konusu ise, akla gelmesi gereken ilk şey, bu nesnelere bağlı olan referansın gösterdiği hedeflerde bir değişiklik olacağıdır.

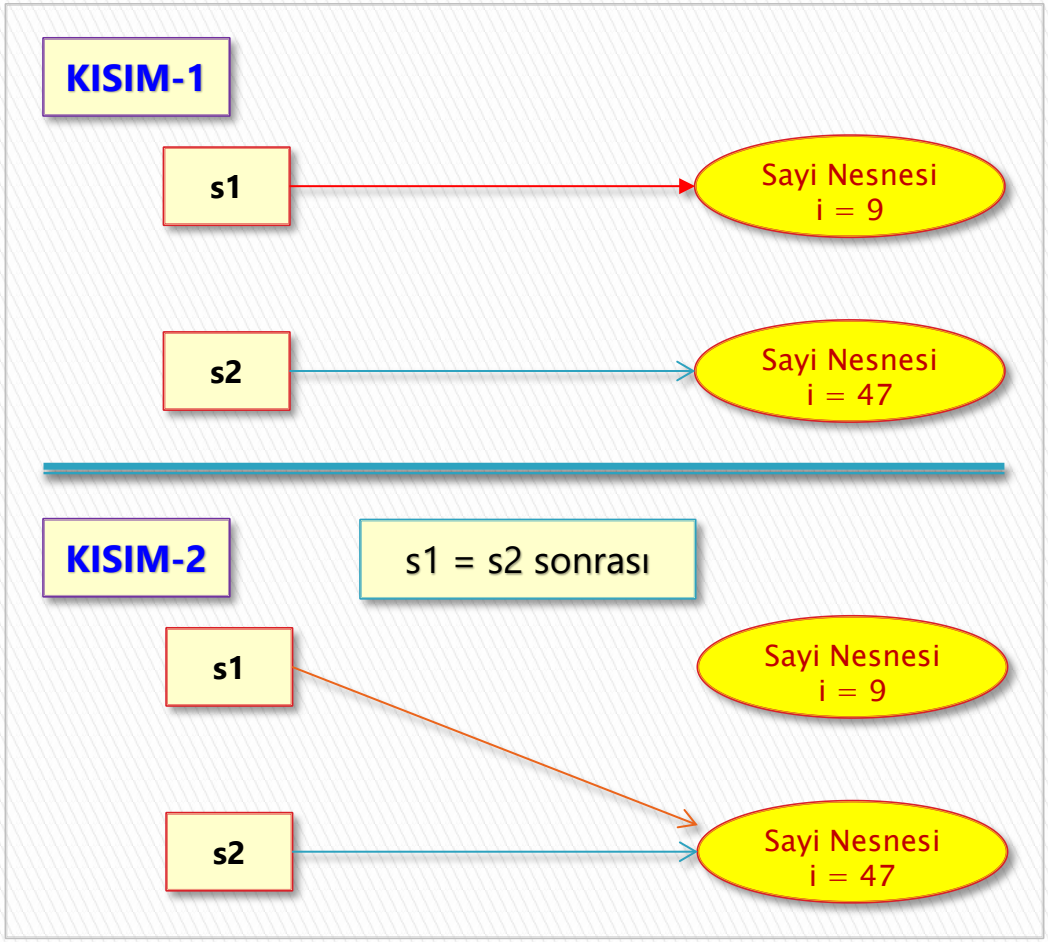
```
class Sayi {
  int i;
}

public class NesneAtama {
  public static void main(String[] args) {
    Sayi s1 = new Sayi();
    Sayi s2 = new Sayi();
    s1.i = 9;
    s2.i = 47;
    System.out.println("1: s1.i: " + s1.i + ", s2.i: " + s2.i);
    s1 = s2; // Referanslar kopyalaniyor.. Nesneler degil
    System.out.println("2: s1.i: " + s1.i + ", s2.i: " + s2.i);
    s1.i = 27;
    System.out.println("3: s1.i: " + s1.i + ", s2.i: " + s2.i);
  }
}
```

```
1: s1.i: 9,  s2.i: 47
2: s1.i: 47, s2.i: 47
3: s1.i: 27, s2.i: 27
```

Yukarıda verilen uygulamada, iki adet nesne oluşturulmaktadır. Bunlar, Sayi tipindeki **s1** ve **s2** referanslarına bağlanmışlardır.

Nesne Referans Değişikliği



s1 referansının daha önceden işaret etmiş olduğu *Sayi* nesnesine ne olacaktır?

Bu nesne kullanılmayacağından dolayı çöp haline gelecektir ve çöp toplayıcısı (*Garbage Collector*) tarafından temizlenecektir.

Nesnelerin Karşılaştırılması

```
public class CompareObjs {
  public static void main(String[] args) {
    String str1 = new String("Yeni bir metin");
    String str2 = new String("Yeni bir metin");
    System.out.println(str1 == str2); // false
    System.out.println(str1.equals(str2)); // true
  }
}
```

```
false
true
```

Nesnelerin yerine, temel (*primitive*) tiplere ait değerler ve Sarmalayıcı sınıf değerleri için == operatörü ile num1 ve num2 değişkenlerinin değerleri karşılaştırıldığında;

```
public class CompareVals {
  public static void main(String[] args) {
    Integer num1 = 47;
    Integer num2 = 47;
    System.out.println(num1 == num2); // true
    System.out.println(num1.equals(num2)); // true
  }
}
```

```
true
true
```

String Nesnesi ve String Pool

- **new()** operatörü ile oluşturulan bütün String objeleri yeni bir nesne olarak heap'te yerini alır.

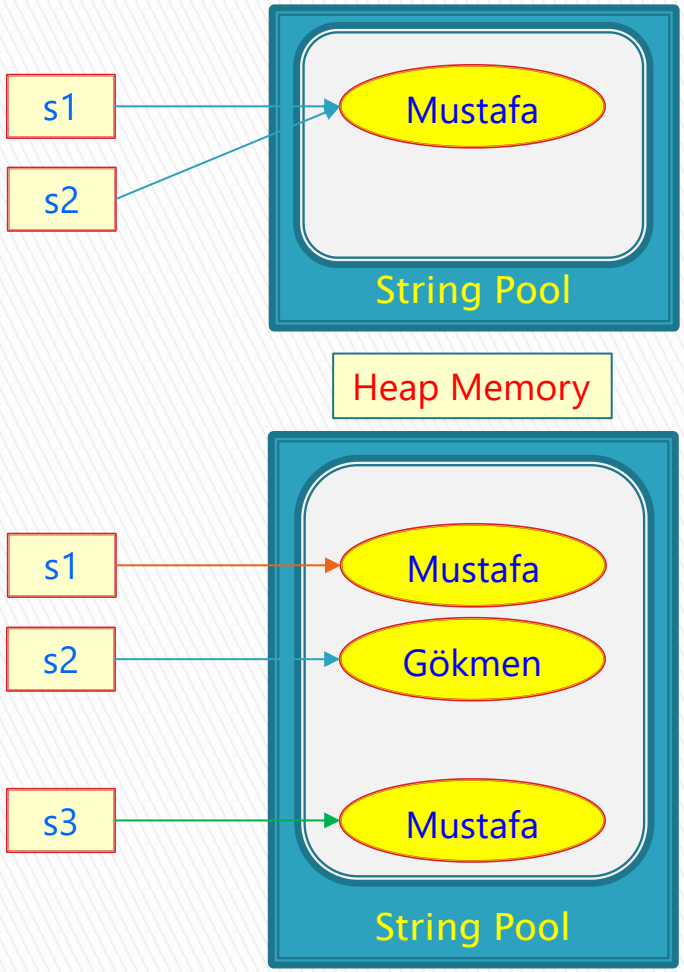
```
String s1 = new String("Mustafa");
```

- Bundan sonra `String s2 = "Mustafa";` şeklinde yeni bir nesne üretilirse, bu nesnenin değeri String Pool'da varsa yeni bir nesne üretilmez ve var olan önceki nesnenin adresi döndürülür.

```
String s2 = "Mustafa";
```

- "=" operatörü ile her String nesnesi üretildiğinde JVM ilk olarak String Pool'u kontrol eder. Eğer oluşturulan String nesnesi pool'da varsa yeni bir nesne oluşturulmaz ve bu nesnenin adresi döndürülür ve eğer yoksa pool'da yeni bir String objesi oluşturulur.

```
String s1 = "Mustafa";
String s2 = "Gökmen";
String s3 = new String("Mustafa");
```



String Nesnesi ve String Pool

```java
public class StringPool {
  public static void main(String[] args) {
    String s1 = new String("Teknoloji");
    String s2 = new String("Tekno");
    String s3 = new String("loji");
    String s4 = s2 + s3;
    String s5 = "Tek" + "noloji";

    System.out.println(s1 == s4);      // false
    System.out.println(s1.equals(s4)); // true
    System.out.println(s1 == s5);      // false
    System.out.println(s1.equals(s5)); // true
  }
}
```

```
false
true
false
true
```

- ✓ s4 ve s5 değerleri çalışma zamanında hesaplanarak,
- ✓ diğer değerler derleme zamanında atanarak belirleniyor.

Öğr.Gör. Mustafa GÖKMEN
gokmen@selcuk.edu.tr

Nesneye Yönelik Programlama

Type Casting

Tip Dönüşümleri (Type Casting)

- Operatörler ile ilgili işlemler yapılırken dikkat edilmesi gereken önemli konulardan biri tip dönüşümleri (type casting) konusudur. İki farklı veri tipinin operasyonu sırasında operandlardan birinin veri tipinin dönüştürülmesi işlemine tip dönüşümü (type casting) denir.

- Her iki (*Implicit* & *Explicit*) tip dönüşümünde de bilinmesi gereken,
  - ✓ *küçük kapasiteli tiplerden büyük kapasiteli türlere*

  doğru dönüşüm yapılması gerektiğidir. Aksi halde veri kaybı olacaktır.

```
int a;
short b;
b = (short) a;  // Hatalı tip dönüşümü.
a = (int) b;    // Veri kaybı yok.
```

Tip Dönüşümleri (Type Casting)

```
public class TypeCast1 {
  public static void main(String[] args) {
    int a = 5;
    int b;
    double c = 18.6;

    // Implicit (Örtülü) tip dönüşümleri
    b = a / 2; // Automatic Conversion ile veri kaybı
    System.out.println("b -> " + b); // b -> 2 (Veri Kaybı)

    a = c; // Invalid Casting (HATA!: Derlenemez)
    System.out.println("a -> " + a); // a -> Error

    // Explicit (Doğrudan) tip dönüşümleri
    b = (int) c; // Veri kaybı (Ondalık kısım yok edildi)

    System.out.println("b -> " + b); // b -> 18
    c = (double) a / 2; // Doğru ve Geçerli
    System.out.println("c -> " + c); // b -> 2.5
  }
}
```

Nümerik tip dönüşüm örnekleri

Tip Dönüşümleri (Type Casting)

```
public class TypeCast2 {
    public static void main(String[] args) {
        int a = 5;
        double b = (long) a; // ???
        double x = 4.15;
        int y = (int) x;
        float z = (float) x; // ???

        System.out.println("b = " + b + ", y = " + y + ", z = " + z);
    }
}
```

```
b = 5.0, y = 4, z = 4.15
```

String'den `int` tipine dönüşüm için `Integer` sınıfının iki metodunu kullanabiliriz.

```
String degis = "5";
int yeni1 = Integer.valueOf(degis);
int yeni2 = Integer.parseInt(degis);
```

Bunun tersi de mümkündür (`int` türünden `String` türüne):

```
int degis = 42;
String yeni1 = String.valueOf(degis);
String yeni2 = Integer.toString(degis);
```

Tip Dönüşümleri (Type Casting)

KAYNAK TİP	OLASI HEDEF TİPLER
byte	short, int, long, float, double
short	int, long, float, double
int	long, float, double
long	float, double
float	double

Casting : byte -> short -> int -> long -> float -> double

```
int a = 5, b;
double c;

// Implicit (Dolaylı) tip dönüşümü
b = a / 2; // Sonuc bölümden sonraki tamsayı kısmı
System.out.println("b -> " + b); // c -> 2

// Explicit (Doğrudan) tip dönüşümü
c = (double) a / 2; // a, bölmeden önce double tipe dönüştürülünce
                    // bölümün sonucu da double tipinde olacaktır.
System.out.printf("c -> %f", c); // b -> 2,500000
```